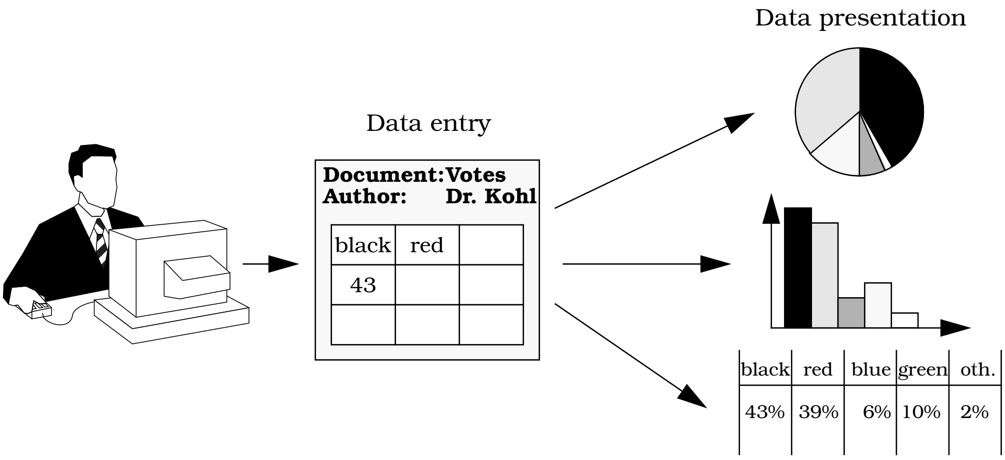
Presentation-Abstraction-Control

The *Presentation-Abstraction-Control* architectural pattern (PAC) defines a structure for interactive software systems in form of a hierarchy of cooperating agents. Every agent is responsible for a specific aspect of the application functionality and consists of three components: presentation, abstraction, and control. This subdivision separates the human-computer interaction aspect of the agent from its functional core and its communication with other agents.

Example Consider a simple information system for political elections. It offers a spreadsheet for entering data and several kinds of tables and charts for presenting current standings. Users can interact with the software through a graphical interface.



Different versions, however, adapt the user interface to specific needs. For example, one version will support additional views on the data, such as the assignment of parliament seats to political parties. The software should also support multi-tasking. Data entry personnel, for instance, enter new data while another user presents the latest projection.

Context Developing an interactive application with help of specialized agents[1].

Problem Interactive systems often can be viewed as a set of cooperating agents. Agents specialized in human-computer interaction accept user input and visualize data. Other agents maintain the data model of the system and offer functionality which operates on these data. Additional agents are responsible for diverse tasks such as error handling or the communication with other software systems. Besides this vertical decomposition of system functionality, we often encounter a horizontal decomposition. Production planning systems (PPS), for example, distinguish between production planning and the execution of a previously specified production plan. For each of these tasks separate agents can be defined.

In such an architecture of cooperating agents each agent is specialized for a specific task and all agents together provide the system functionality. This architecture also captures both a vertical and horizontal decomposition. The following *forces* effect the solution:

- Agents often maintain their own state and data. For example, in a PPS system, the production planning and the actual production control may work on different data models, one tuned for planning and simulation and one which is performance-optimized for efficient production.
- Interactive agents provide their own user interface since their respective human-computer interactions are often widely different. For example, entering data into spreadsheets is based on keyboard input, while manipulating graphical objects uses a pointing device.
- Cooperating agents need a mechanism to effectively exchange data, messages, and events.
- Many large applications—especially interactive ones—are multi-user systems. Thus, multi-tasking is a major concern when designing such software systems.

---

1. In the context of this pattern an *agent* denotes an information processing component which includes event receivers and transmitters, data structures to maintain state, and a processor that processes incoming events, updates its own state, and may produce new events [BaCo91]. Also, we use the terms *agent* and *PAC agent* as synonyms in this pattern description.

- Systems evolve over time. Especially the presentation aspect is prone to change. Using graphics, and more recently, including multimedia features, are examples of pervasive changes to user interfaces. Changes to individual agents or extending the system with new agents, however, should not affect the whole system.

Solution Structure the interactive application as a hierarchy of so called *PAC agents*. There is one top-level PAC agent, several intermediate-level PAC agents, and even more bottom-level PAC agents. Every agent is responsible for a specific aspect of the application functionality and consists of three components: presentation, abstraction, and control.

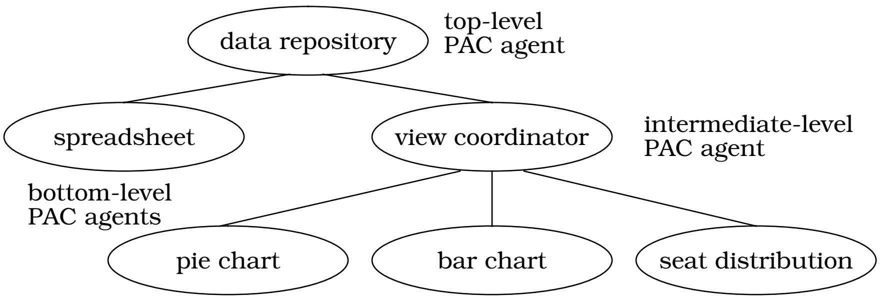
The *presentation* component provides the perceivable behavior of the PAC agent. The *abstraction* component maintains the data model underlying the agent and provides functionality which operates on these data. The *control* component connects the presentation and abstraction components and provides functionality which allows the agent to communicate with other PAC agents.

The *top-level PAC agent* provides the functional core of the system. Most other PAC agents depend or operate on this core. Furthermore, the top-level PAC agent includes those parts of the user interface which cannot be assigned to particular subtasks, such as menu bars or a dialog box with information about the application.

Bottom-level PAC agents represent self-contained semantic concepts on which users of the system can act, for example spreadsheets and charts. The bottom-level agents present these concepts to the user and provide all operations users can perform on these agents, such as zooming or moving a bar chart.

Intermediate-level PAC agents represent either combinations of or relationships between lower-level agents. For example, an intermediate-level PAC agent may maintain several views onto the same data, such as a floor plan and an external view of a house in a CAD system for building architecture.

➥ In our information system for political elections we define a top-level PAC agent which includes the data repository of the system. It also provides the functionality to calculate percentage shares and seat distributions. At the bottom-level we specify four PAC agents: one spreadsheet agent for entering data and three view agents for each kind of diagram representing the data. The application has one intermediate-level PAC agent. It coordinates the three bottom-level view agents and keeps them consistent. The spreadsheet agent is directly connected to the top-level PAC agent. Users of the system only interact with bottom-level agents. ❑



Structure The main responsibility of the *top-level PAC agent* is to provide the global data model of the software. It is maintained in the abstraction component of this agent. The interface of the abstraction offers functions to manipulate the data model and to retrieve information about it. The representation of data within the abstraction component is media-independent. For example, in a CAD system for building architecture walls, doors, and windows are represented in centimeters or inches reflecting their real size, not in pixels for display purposes. This supports an adaptation of the PAC agent to different environments without major changes in its abstraction component.

The top-level presentation component often has little responsibility. It may include user interface elements which are common to the whole application. In some systems, such as the network traffic manager [TS93], there is no top-level presentation at all.

The control component of the top-level PAC agent has three responsibilities. First, it allows lower-level agents to make use of the top-level services, mostly to access and manipulate the global data model. Incoming service requests from lower-level agents are forwarded either to the abstraction or to the representation component.

Second, the control coordinates the hierarchy of PAC agents. It maintains information about connections of the top-level agent to lower-level agents. The control uses this information to ensure correct collaboration and data exchange between the top-level agent and lower-level agents. It encompasses dedicated functions for exchanging events, messages, and data with lower-level agents.

Third, the control maintains information about the interaction of the user with the system. For example, it may check whether a particular operation on the data model triggered by the user can be performed. Or the control keeps track of the functions called for providing history or undo/redo services for operations on the functional core.

➥ In our example, the abstraction component of the top-level PAC agent maintains the data repository underlying the system. It also implements all functions which operate on this repository, such as algorithms for calculating projections and seat distributions. It further includes functions to maintain data, such as for updating and consistency checking. The control component organizes the communication and cooperation with lower-level agents, namely the view coordinator and spreadsheet agents. This top-level PAC agent does not include a presentation component. ❏

Bottom-level PAC agents represent specific semantic concept of the application domain, for example a mailbox in the traffic management system [TS93] or a wall in the mobile robot system [Cro85]. This semantic concept may be as low-level as a simple graphical object like a circle or as complex as a bar chart summarizing all data in the system.

The presentation component of a bottom-level PAC agent presents a specific view onto the corresponding semantic concept and provides access to all functions users can apply on it. Internally, the presentation component also maintains information about the view, for example its position on the screen.

The abstraction component of a bottom-level PAC agent has a similar responsibility as the abstraction component of the top-level PAC agent. It maintains agent specific data. In contrast to the top-level abstraction, however, no other PAC agents depend on these data.

The control component of a bottom-level PAC agent maintains consistency between the abstraction and presentation component, thereby avoiding direct dependencies between them. It serves as an adapter and performs both interface and data adaptation. Furthermore, the control communicates with higher level agents to exchange data, events, or status information. Incoming events, messages, or data from higher level PAC agents are forwarded either to the abstraction or to the presentation component.

Concepts represented by bottom-level PAC agents, such as our bar and pie charts, are atomic in the sense that they are the smallest units a user can manipulate. For the election system this means that users can only operate on the various charts with election data as a whole, for instance by changing the scaling factor of the y-axis. They cannot, for example, resize an individual bar of a bar chart.

Bottom-level PAC agents are not restricted to provide semantic concepts of the application domain. You can also specify bottom-level agents which implement system services. For example, there may be a communication agent which allows the system to cooperate with other applications and to monitor this cooperation.

➥ Consider a bar chart agent in our example system. Its abstraction component caches the election data presented in the chart and maintains chart-specific information such as their order of presentation. The presentation component is responsible for presenting the bar chart in a window and for providing all functions which can be applied on it, such as zooming, moving, and printing the chart. The control component serves as a level of indirection between its presentation and abstraction component. The control is also responsible for the communication with the view coordinator agent.❑

Intermediate-level PAC agents can embody two different roles: composition and coordination. When, for instance, each object in a complex graphics is represented by a separate PAC agent, an intermediate-level agent composes these objects to a composite graphical object. The intermediate-level agent defines a new abstraction, whose behavior encompasses both the behavior of its components and the new characteristics that are added to the composite. The second role of an intermediate-level agent is to maintain consistency between lower-level agents, for example when coordinating multiple views onto the same data.

The abstraction component maintains the specific data of the intermediate-level PAC agent. The presentation component implements its user interface. The control component has all responsibilities of the control components of bottom-level PAC agents and the top-level PAC agent.

➥ For our example system, we define one intermediate-level PAC agent. Its presentation component provides a palette which allows users to create views onto the election data, such as bar or pie charts. The abstraction component maintains data about all currently active views, each of which is realized by its own bottom-level agent. The main responsibility of the control component is to coordinate all subordinate agents. It forwards incoming notifications about data model changes in the top-level agent to the bottom-level agents and organizes their update. It also includes functionality to create and delete these agents upon user request. ❏

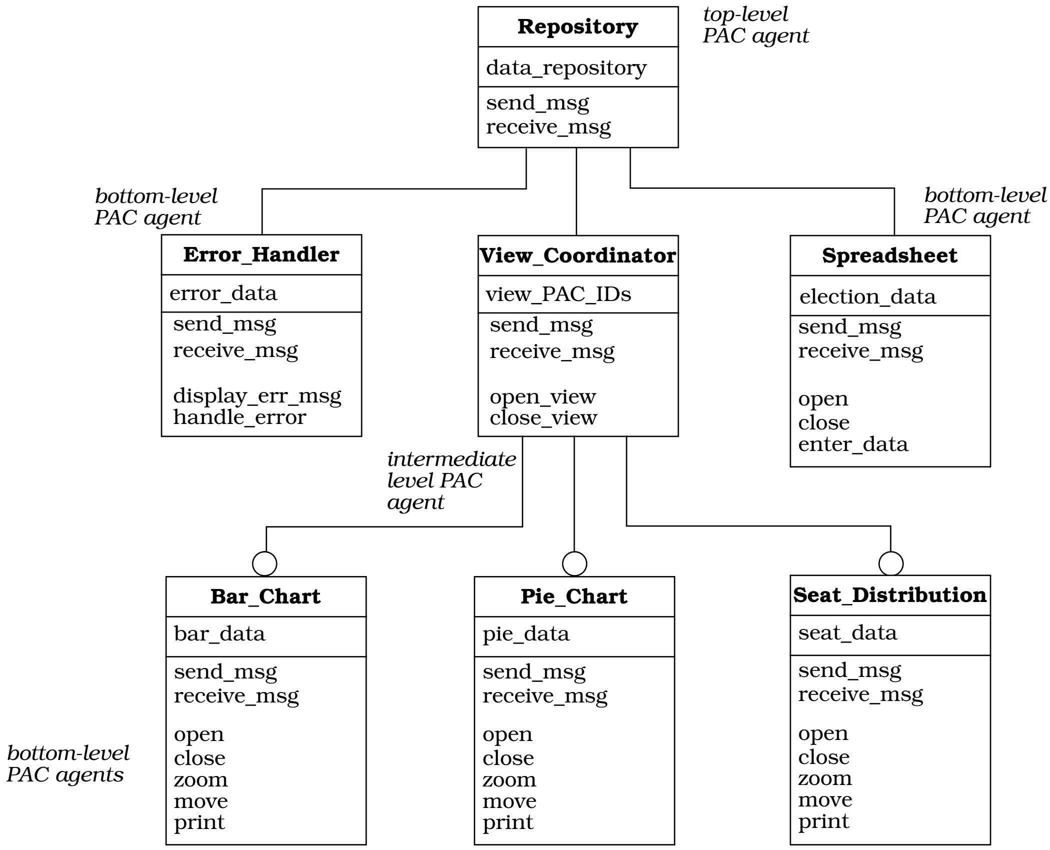
Class Top-Level Agent	***Collaborators***
Responsibility • Provides the functional core of the system. • Controls the PAC hierarchy.	• Intermediate-Level Agents • Bottom-Level Agents

Class Interm.-Level Agent	***Collaborators***
Responsibility • Coordinates lower-level PAC agents. • Composes lower-level PAC agents to a single unit of higher abstraction.	• Top-Level Agent • Bottom-Level Agents

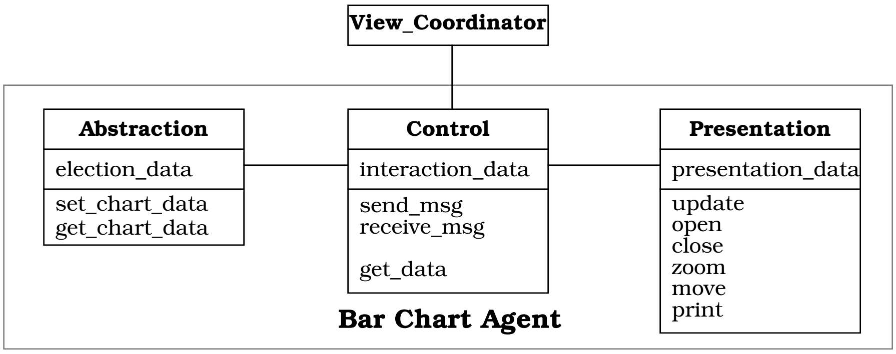
Class Bottom-Level Agent	***Collaborators***
Responsibility • Provides a specific view onto the software or a system service, including its associated human-computer interaction.	• Top-Level Agent • Intermediate-Level Agents

The following diagram illustrates the PAC hierarchy of our information system for political elections. However, it only lists those functions which are necessary for controlling and coordinating the

PAC hierarchy or which are accessible to other PAC agents or the user.



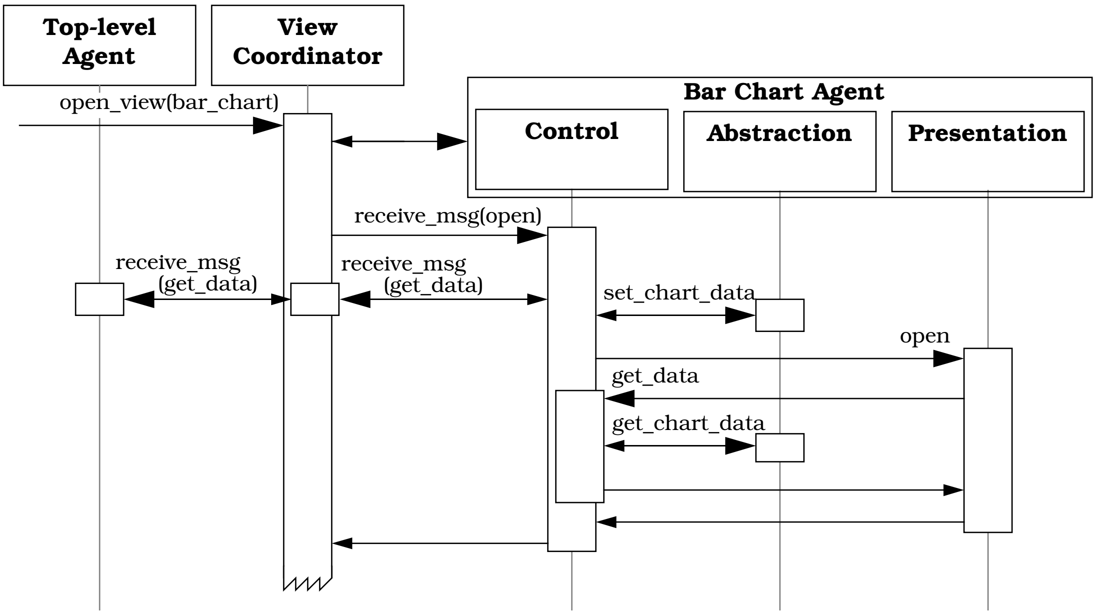
The internal structure of a PAC agent is illustrated with help of the bar chart agent from our example:



Dynamics We will illustrate the behavior of a PAC architecture with two scenarios, both based on our motivating example.

Scenario I describes the cooperation between different PAC agents when opening a new bar chart view onto the election data. The scenario also includes a more detailed view on the internal behavior of the bar chart agent. It is divided into five phases:

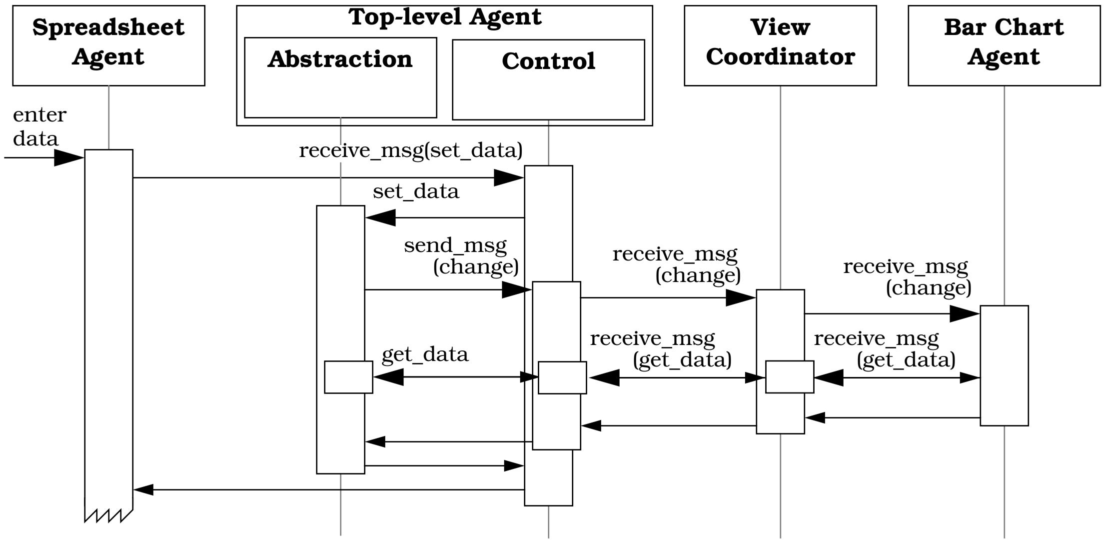
- A user asks the presentation component of the view coordinator agent to open a new bar chart.
- The control of the view coordinator agent instantiates the desired bar chart agent.
- The view coordinator agent sends an open event to the control of the new bar chart agent.
- The control of the bar chart agent first retrieves the data from the top-level PAC agent. The view coordinator agent mediates between bottom- and top-level. The data returned to the bar chart are stored in its abstraction component. Then, its control calls the presentation component to display the chart.
- The presentation component pops up a new window on the screen, retrieves the data from the abstraction component by asking the control, and finally displays them within the new window.



There are obvious optimizations in here like caching top-level data in the view coordinator or first calling the bottom-level presentation component and then storing the data. But, at this point, our emphasis is on explaining the basic ideas of the pattern.

Scenario II shows the behavior of the system after entering new election data, with a closer look on the internal behavior of the top-level PAC agent. It has five phases.

- The user enters new data into a spreadsheet. The control of the spreadsheet agent forwards these data to the top-level PAC agent.
- The control of the top-level PAC agent receives the data and causes the top-level abstraction to change its data repository. The top-level abstraction asks its control to update all agents that depend on the new data. Hence, the control of the top-level PAC agent notifies the view coordinator agent.
- The control of the view coordinator agent forwards the change notification to all view PAC agents it coordinates.
- Similarly to the previous scenario, all view PAC agents update their data and refresh the image they display.



Implementation We outline the implementation of a PAC architecture in ten steps. Iterate through any subsequence, if necessary.

1 *Define a model of the application.* Analyze the problem domain and map it onto an appropriate software structure. Do not consider the distribution of components to PAC agents when performing this step. Concentrate on finding a proper decomposition and organization of the application domain. To this end answer the following questions:

- Which services should the system provide?
- Which components can fulfill these services?
- What are the relationships between components?
- How do the components collaborate?
- What data do the components operate on?
- How will the user interact with the software?

Follow an appropriate analysis method when specifying the model.

2 *Define a general strategy for organizing the PAC hierarchy.* At this point, we do not yet have defined individual agents but can specify general guidelines for organizing the upcoming hierarchy of cooperating agents.

One rule to follow is '*lowest common ancestor*'. When a group of lower-level agents depends on the services or data provided by another agent we try to factor this into an agent which will be the root of the subtree formed by the lower-level agents. As a consequence only agents providing global services rise to the top of the hierarchy. All agents in the election system depend on the central data repository. It is therefore provided by the top-level PAC agent. If only a fraction of all agents would depend on the repository we would try to group them into a subtree and come up with an agent holding the repository at the root of that subtree.

A second aspect to consider is the depth of the hierarchy. Most PAC architectures comprise several intermediate levels of PAC agents. In the mobile robot system, for example, bottom-level agents are composed to environments which again are composed to workspaces [Cro85]. The deeper the hierarchy is the better it often reflects the decomposition of the application into self-contained concepts. On the other hand, deep hierarchies tend to be run-time inefficient and hard to maintain. Finding the appropriate decomposition of a system into PAC agents is important to gain the benefits of this architecture.

3 *Specify the top-level PAC agent.* Identify those parts of the analysis model which represent the functional core of the system. These are mostly components maintaining the global data model of the system and components directly operating on these data. Furthermore, identify all user interface elements which are common for the whole application, such as menu bars or dialogs with information about the system. All components we identify in this step will be part of the top-level agent.

4 *Specify the bottom-level PAC agents.* Identify those components of the analysis model which represent the smallest self-contained units of the system on which the user can perform operations or view presentations. In our example system these units are the various diagrams and charts presenting election data and the spreadsheet for entering these data.

For each of these units identify those components which provide the human-computer interaction associated with them. A bar chart in our example requires a window in which the diagram is displayed and functionality to manipulate the diagram, such as zooming and printing it. Every such semantic concept like a bar chart and its user interface components together will form a separate bottom-level agent.

5 *Specify bottom-level PAC agents for system services.* Often an application includes additional services which are not directly related to its primary subject. In our example system we define an error handler. Other systems may provide services for communicating with other systems or for configuration purposes. Each of these services, including their human-computer interaction, can be implemented as a separate bottom-level agent [BaCo91].

6 *Specify intermediate-level PAC agents to compose lower-level PAC agents.* Often, several lower-level agents together form a higher level semantic concept on which users can operate.

In the mobile robot system described in [Cro85], several wall, place, and route PAC agents form an environment. Users of the system can specify new environments and missions for robots within environments. Furthermore, environments are displayed on the screen, and users perform actions like scrolling and zooming on these presentations. Thus, an environment is a higher level concept with its own functionality and human-computer interaction. Such concepts are implemented as separate agents. They provide their own human-computer interaction and operate on their constituent lower-level agents.

➥ The election example does not provide semantic concepts above individual charts, diagrams, and spreadsheets. Therefore we do not define PAC agents for composing other PAC agents. ❏

7 *Specify intermediate-level PAC agents to coordinate lower-level PAC agents.* Many systems offer multiple views onto the same semantic concept. For example, in text editors you find ‘layout’ and ‘edit’ views onto a text document. When the data in one view change, all other views must be updated. Such coordination components, which may have not been identified when modelling the analysis model, provide their own human-computer interaction, for example menu entries and associated callback functions. The view coordinator agent of our example system is such an intermediate-level agent. To implement agents which coordinate multiple views you may apply the View Handler pattern (167).

Note that not only views must be coordinated. The traffic control management system described in the Known Uses section [TS93], for example, implements an agent which coordinates the different jobs the system currently performs on the telecommunication network.

8 *Separate core functionality from human-computer interaction.* For every PAC agent, introduce presentation and abstraction components. All components which provide the user interface of the agent, such as graphical images presented to the user, presentation-specific data like screen coordinates, or menus, windows, and dialogs form the presentation part. All components which maintain core data or operate on them will form the abstraction.

You can give unified access to the abstraction and presentation components by applying the Facade pattern [GHJV95]. The control exports those parts of the abstraction and presentation interfaces other components can use.

For some PAC agents it may be hard to specify presentation or abstraction parts. For instance, top-level PAC agents often do not provide a presentation component. [BaCo91] suggest to implement the top-level presentation as a general geometry manager which maintains spatial relationships between the presentation parts of lower-level PAC agents. To further organize the presentation part you may apply the Command Processor pattern (143). This enables scheduling user requests for deferred or prioritized execution and to provide agent-specific undo/redo services.

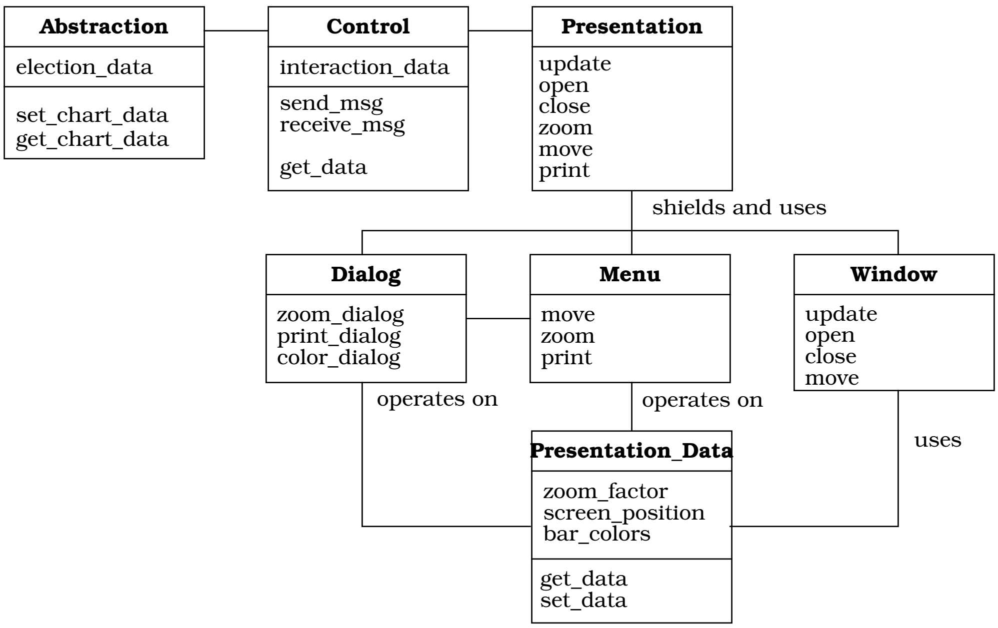
Some abstraction components, especially those in lower-level agents, often operate on data provided by other PAC agents. In this case, you may either not specify an abstraction component, or the abstraction component just serves as a data cache. In the first case, you save all the effort to implement components keeping data replica and the functionality to keep these replica consistent. In the latter, you save additional communication effort between PAC agents, for example when refreshing a view after having moved a window.

Finally, introduce the control component to mediate between the abstraction and presentation part and to avoid direct dependencies between them. The control is implemented as an adapter [GHJV95]. It links the presentation and abstraction together by performing interface and data adaptation between them. In this step, do not consider those parts of the control which deal with the communication of the agent to other PAC agents. This is a different role of the control component and should therefore be separated from the mediation between the agent internal abstraction and presentation components.

➥ To illustrate this step, we refine the bar chart agent from the structure section. The abstraction component keeps a copy of those election data displayed in the bar chart.

The presentation component is structured into components providing the respective functionality of windowing, menus, dialogs, and maintaining presentation-specific data. To shield clients from this structure we provide a facade [GHJV95].

The control of the pie chart PAC agent is simple. It just forwards data reading requests from the presentation to the abstraction component. Communication with higher-level agents is handled in the next step.❑



9 *Provide the external interface.* To cooperate with other agents every PAC agent sends and receives messages, events, state information, and data. Implement this functionionality as part of the control component.

Within an agent, incoming messages, events, or data are forwarded to their intended addressee. The addressee may be the abstraction or the presentation component of the agent but also lower- or higher-level agents. For example, the view coordinator agent of our information system regularly receives change notifications from the top-level PAC agent and forwards them to the view agents. It also receives service requests from lower-level agents which are forwarded to the top-level agent. In other words, the control is a mediator, and you may use the Mediator pattern [GHJV95] to implement this role. Use the Composite Message pattern [SC95] to support flexible data exchange between PAC agents, specifically to be independent of particular data formats, marshalling, and unmarshalling, fragmentation, and re-assembling methods.

A PAC agent can be connected with other PAC agents in a flexible and dynamic way by using registration functionality. For example, if a new instance of the bar chart agent in our election system is created, it is dynamically registered with the view coordinator agent.

If a PAC agent depends on data or information maintained by other PAC agents, provide a change propagation mechanism. Such a mechanism involves all agents and all levels of the hierarchy and works in both directions. Upon changes to data within an agent, its abstraction component starts the change propagation. The control component forwards change notifications to all dependent PAC agents, but often also to the presentation component. Incoming change notifications from other agents cause the abstraction and presentation components to update their internal states. One way to implement the change propagation mechanism is to use the Publisher-Subscriber pattern (151). Another way is to integrate the change propagation with the general functionality for sending and receiving events, messages, and data, see below.

The interface for these communication and cooperation functions should be the same for all PAC agents. This supports re-configuration and reuse of PAC agents, and the extension of the application with new PAC agents.

➥ For the control of the view coordinator PAC agent we provide the following interface:

```
enum ViewKind { bar_chart, pie_chart, seat_distribution };
    // Enumeration type to denote available views onto
    // election data

class DataSetInterface { // ... };
    // Common interface for datasets, messages, and
    // events, according to the specifications of the
    // Composite Message pattern [SC95]

class PAC_Id { // ... };
    // Provides a handle to a PAC agent

class VC_Control {
    // Data member specifications
    PAC_Id        parent;  // higher level agent
    List<PAC_Id>  children;// lower-level agents
    // More data member specifications ...
private:
    void attach(PAC_Id agent, parent_agent = 0);
    void detach(PAC_Id agent);
        // Registration functionality for connecting
        // dependent view agents and the top-level agent
        // with the view coordinator agent.

    DataSetInterface send_msg(DataSetInterface data);
        // Sending events, messages, or data
        // to other PAC agents including
        // change notifications

    void open_view(ViewKind kind);
    void close_view(PAC_Id agent);
        // Opening and closing views including
        // creation, registration,and deletion
        // of bottom-level agents displaying charts
public:
    DataSetInterface receive_msg(DataSetInterface data);
        // Receiving events, messages, or data
        // from other PAC agents including
        // change notifications
};
```

`send_msg` and `receive_msg` return objects for hold ing answers. ❑

10 *Link the hierarchy together*. After implementing the individual PAC agents we can build the final PAC hierarchy. Connect every PAC agent with those lower-level PAC agents with which it directly cooperates.

Provide those PAC agents which dynamically create and delete lower-level PAC agents with functionality to dynamically extend or reduce the PAC hierarchy. For example, the view coordinator agent in our information system creates a new view PAC agent if the user wants to open a particular view and deletes this agent when the user closes the window in which the view is displayed.

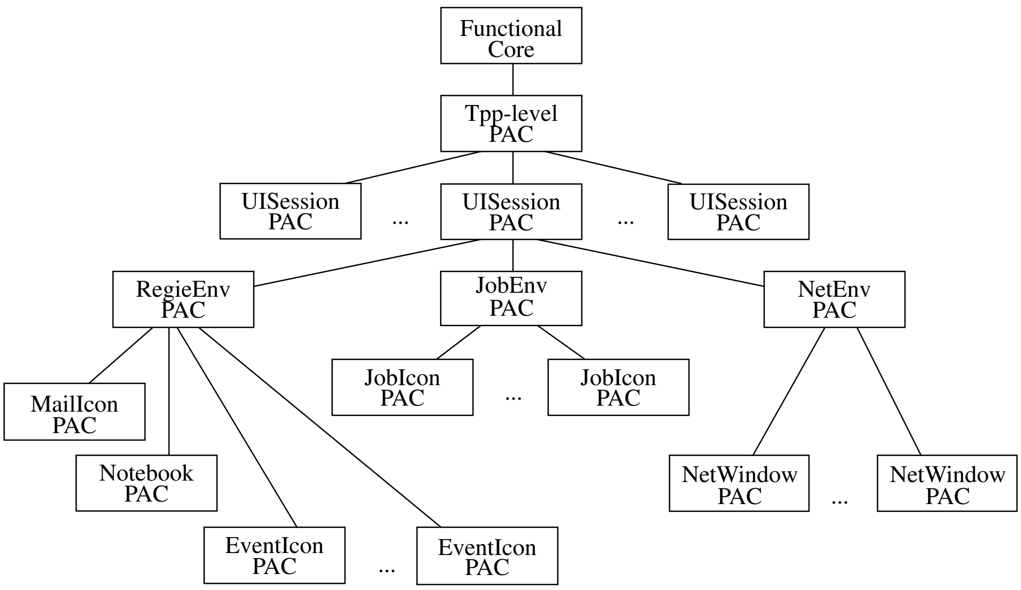
Variants *PAC agents as active objects*. Many applications, especially interactive ones, benefit from multi-threading. The mobile robot system [Cro85] is an example of a multi-threaded PAC architecture. Every PAC agent can be implemented as an active object which lives in its own thread of control. Design patterns like Active Object and Half-Sync/Half-Async [Sch95] help with implementing such an architecture.

PAC agents as processes. To support PAC agents located in different processes or on remote machines use proxies (145) to locally represent these PAC agents and to avoid direct dependencies on their physical location. Use the Forwarder-Receiver pattern (147) or the Client-Dispatcher-Server pattern (149) to implement the interprocess communication between PAC agents.

Known Uses **Traffic Control Management**. This system is described in [TS93]. It visualizes the traffic in telecommunication networks. Every 15 minutes all monitored switching units report their current traffic situation to a control point where the data are stored, analyzed, and visualized. This helps with recognition of potential bottlenecks and preventing traffic overload. The system includes functions for:

- gathering traffic data from switching units,
- threshold checking and generation of overflow exceptions,
- logging and routing of network exceptions,
- visualization of traffic flow and network exceptions,
- various user configurable views onto the whole network,
- statistical evaluations of traffic data,
- access to historic traffic data,
- system administration and configuration.

The design and implementation of the system follows the Presentation-Abstraction-Control pattern. Every function of the system is represented by its own bottom-level PAC agent. There are dedicated agents for every single view on the network, for the jobs the system can perform, and for the additional services the system offers, such as mail or help. Three intermediate-level PAC agents coordinate these bottom-level PAC agents, one for each of the above three categories view (NetEnv in the picture), jobs (JobEnv), and additional services (RegieEnv). An additional intermediate PAC agent organizes user sessions. The top-level PAC agent coordinates individual user sessions and communicates with the functional core of the system which is implemented separately from the PAC hierarchy, probably due to incorporating legacy software. The PAC agent hierarchy of the system is dynamic. If, for example, a user starts a new session, a corresponding UISession agent is created and registered with the top-level agent. At the end of the session this agent is deleted again.



Mobile Robot. This system [Cro85] allows an operator to interact with a mobile robot which navigates within a closed hazardous environment. Such an environment consists of walls, equipment, and people, either intruders or accident victims. The robot navigates in this environment using its own sensors and information from the system operator. The software allows to

- provide the robot with a description of the environment it will work in next, places in this environment, and routes between places,
- subsequently modify this environment,
- specify missions for the robot,
- control the execution of missions,
- observe the progress of missions.

Each wall, route, and place within an environment is represented by its own bottom-level PAC agent. These agents together visualize the environment. Environments are represented by intermediate-level PAC agents. They control the constituent wall, route, and place PAC agents. The functionality users can perform on an environment is implemented in a so-called palette PAC agent, which is also at the bottom-level of the hierarchy. The environment PAC agent and the palette PAC agent form a workspace for the robot. This workspace is represented by its own intermediate-level PAC agent. To support multiple views onto the same environment, a multi workspace PAC agent coordinates all different views onto the same workspace. The PAC agent at the top-level of the hierarchy encapsulates the functional core of the application, which is a rule based intelligent supervisor for navigating and controlling the robot.

Consequences The Presentation-Abstraction-Control architectural pattern has several **advantages**:

Separation of concerns: Different semantic concepts of the application domain are represented by separate agents. Each agent maintains its own state and data, coordinated with, but independent of other PAC agents. Individual PAC agents also provide their own human-computer interaction. This allows to develop a dedicated data model and user interface for every semantic concept or task within the application, independently of other semantic concepts or tasks.

Support for multi-tasking: PAC agents can be easily distributed to different threads, processes, or machines. Extending a PAC agent with appropriate IPC functionality just affects its control component.

Multi-tasking also facilitates multi-user applications. For example, in our information system a TV-moderator can present the latest projection while data entry personnel updates the data base with new election data. Only, the shared data repository or rather its control component has to take care of serialization or synchronisation.

Support for change and extension: Changes within the presentation or abstraction components of a PAC agent do not affect other agents of the system. This allows us to individually modify or tune the data model underlying a PAC agent or to change its user interface, for example stepping from command shells to menus and dialogs.

Also, new agents are easily integrated into an existing PAC architecture without major changes to existing PAC agents. All PAC agents communicate with each other through a pre-defined interface. In addition, existing agents can dynamically register new PAC agents for ensuring communication and cooperation. To add, for example, a new view PAC agent to our information system for political elections, we just need to extend the presentation of the view coordinator PAC agent with an appropriate palette field which allows users to create this new view. The functionality for handling this new PAC agent, for registering it with the view coordinator PAC agent, and for propagating changes and events to it is already available.

The **liabilities** of this pattern are as follows:

Increased system complexity: The implementation of every semantic concept within an application as its own PAC agent may result in a complex system structure. For example, if every graphical object like a circle or square within a graphics editor is implemented as its own PAC agent, the system would drown in a sea of agents. These agents must also be coordinated and controlled which requires additional coordination agents. Think carefully about the level of granularity where to stop refining agents into even more bottom-level agents.

Complex control component: In a PAC system, the control components are the communication mediators between the abstraction and presentation parts of an agent and between different PAC agents. The quality of the control implementations is therefore crucial for an effective collaboration between agents and therefore for the overall quality of the system architecture. All individual roles of control components should be strongly separated from each other. The implementation of these roles should not depend on specific details of other agents, such as their concrete name or physical location in a distributed system. Furthermore, the interface of the control components should be independent of internal details to have collaborators of an agent not depend on the specific interface of presentation or abstraction component of the agent. It is the responsibility of the control to perform any necessary interface and data adaptation.

Efficiency: The overhead in the communication between PAC agents may impact system efficiency. For example, if a bottom-level agent retrieves data from the top-level agent, all intermediate-level agents along the path from the bottom to the top of the PAC hierarchy are involved in this data exchange. If agents are distributed, data transfer also requires IPC, together with marshalling, unmarshalling, fragmentation and re-assembling of data.

These serious potential pitfalls motivate the following discussion about when to use this pattern and when not to use it.

Applicability: The smaller the atomic semantic concepts of an application are and the more similar their user interfaces are, the less applicable this pattern is. For example, a graphical editor where every individual object in a document is represented by its own PAC agent will probably result in a complex fine-grain structure which is also hard to maintain. On the other hand, if the atomic semantic concepts are substantially larger and require their own human-computer interaction, PAC provides a maintainable and extensible structure with clear separation of concerns between different system tasks.

See also The *Model-View-Controller* pattern (63) also separates the functional core of a software system from information display and receiving user input. MVC, however, defines its controller as the entity responsible for accepting user input and translating it into internal semantics. This means that MVC effectively divides the user accessible parts (the presentation in PAC) into view and control. It lacks mediating control components.

Credits PAC was originally described in [Cou87] by Joelle Coutaz. One of the first systems implemented according to PAC was the mobile robot application [Cro85]. Further valuable guidelines for implementing PAC can be found in [BaCo91] and [CNS95]. We thank Joelle Coutaz and Laurence Nigay for fruitful discussions and valuable input which helped us to shape the description of this pattern.